RasPi

DESIGN
BUILD
CODE

Get hands-on with your Raspberry Pi

PREDICT THE WEATHER WITH PI

+ MAKE MINECRAFT CHAT BOT

Plus Master the GPIO Zero Library

Welcome

It's January so it's pretty easy to predict the weather for the next few weeks… rain, cold, drizzle, rain. A more capricious climate is just around the corner though, so we thought we'd showcase an ingenious Raspi project that predicts the weather with a touch of the hand. Swipe left to find out how Jenny Hanell, a Masters student in Human Computer Interaction from Stockholm created Sphaera, a magical globe that foretells the future weather in the local area.

There's plenty in the way of tutorials in this issue too, including creating a chat bot for Minecraft, creating a database with Python and learning your way around GPIO pins.

Get inspired

Discover the RasPi community's best projects

Expert advice

Got a question? Get in touch and we'll give you a hand

Easy-to-follow guides

Learn to make and code gadgets with Raspberry Pi

Editor

From the makers of

LinuxUser & Developer

Join the conversation at…

@linuxusermag | Linux User & Developer | linuxuser@futurenet.com

Contents

Create a chat bot in Minecraft on the Raspberry Pi

Program your Raspberry Pi using Python to read a chat script directly from a text file into Minecraft

Using some basic Python code, we're going to read and write directly from text files saved on your computer into your Minecraft world. Steve (or Alex) will be chatting in game from a pre-written script. This simple chat bot could have all kinds of applications – as with all things Minecraft, the limit is your own imagination!

This tutorial is written with Minecraft Pi Edition in mind, but you don't have to be running Minecraft on a Raspberry Pi to follow along. We've put together a little package that will work on any version of Minecraft, so if you would like to run this tutorial on your favourite flavour of desktop Linux, Pi or no Pi, you can do. To allow Python to hook into Minecraft, you'll need to install McPiFoMo (link available in the Project Essentials section) by extracting the contents of the .minecraft directory into ~/home/.minecraft. McPiFoMo includes MCPiPy from MCPiPy.com and Raspberry Jam, developed by Alexander Pruss. Provided you have Python installed, which of course comes installed as standard on most distros, no additional software is required, other than your favourite text editor or Python IDLE.

Python scripts in this tutorial should always be saved in ~/

THE PROJECT ESSENTIALS

McPiFoMo
http://rogerthat.co.uk/McPiFoMo.rar

Block IDs
http://bit.ly/MC-BlockIDs

Angry IP Scanner
http://angryip.org/download/

home/.minecraft/mcpipy/, regardless of whether you're running Minecraft Pi Edition or Linux Minecraft. Be sure to run Minecraft with the 'Forge 1.8' profile included in McPiFoMo for your scripts to run correctly.

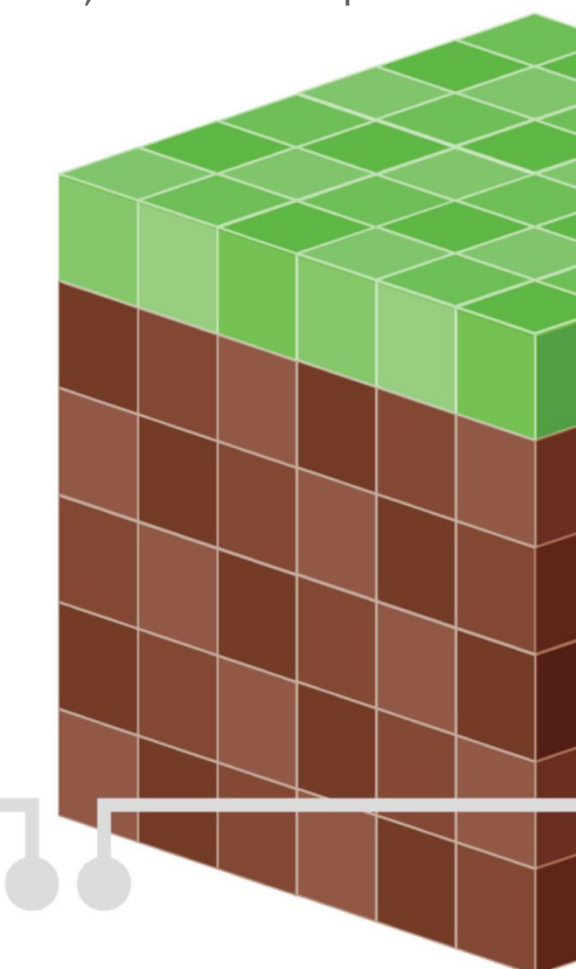
01 Set up your dialogue

We'll want to create two empty documents for this tutorial, one being a regular text file and the other an empty Python script. You can do this in a Terminal window with the following commands:

```
sudo apt-get update
touch ~/.minecraft /mcpipy/script.txt
touch ~/.minecraft /mcpipy/NPC.py
```

Open the newly created files in your favourite text editor. All of this can be done from Terminal, if you use nano, Vim or Emacs.

"Create two empty documents, a regular text file and an empty Python script."



```
File  Edit  View  Search  Terminal  Help
cr@cr-ThinkPad-X131e ~ $ cd ~/.minecraft/mcpipy/
cr@cr-ThinkPad-X131e ~/.minecraft/mcpipy $ touch script.txt
cr@cr-ThinkPad-X131e ~/.minecraft/mcpipy $ touch NPC.py
cr@cr-ThinkPad-X131e ~/.minecraft/mcpipy $ nano NPC.py
```

02 Program your script

Start off the code in your NPC.py file by importing the Minecraft libraries into Python, initiating an instance of Minecraft within a variable (mc) and opening the text file in read-only mode ("r") within another variable (mcScript).

```
from mc import *
mc = Minecraft()
mcScript = open("script.txt", "r")
```

"Import the Minecraft libraries into Python, initiating an instance of Minecraft within another variable"

03 Read, write and append text files in Python

If you would like to call from a text file outside of the mcpipy directory where your Python file is located, you can do so with:

```
mcScript = open("~/.minecraft/mcpipy/script.txt", "r+")
```

You'll also notice we're using "r+" instead of "r" this time. The available modes for editing text files are: append (a), read (r), and read+write (r+).

04 Write to your script with Python

As with any programming tutorial, we start off by printing 'Hello World!':

```
   mcScript = open("~/.minecraft/mcpipy/script.
txt", "r+")
  mcScript.write("Hello World")
  mcScript.close()
```

Here we're opening the file in read+write mode, saving a string to the file and closing it. It's important to note that text files will only save if closed.

05 Display your script in-game

Now that we've got something written in our text file, we can 'print' our scripts in-game:

```
   mcScript = open("~/.minecraft/mcpipy/script.
txt", "r+")
  mc.postToChat(mcScript)
  mcScript.close()
```

Note that we're using the postToChat action from our mc instance, rather than print, and we're posting the contents of a variable instead of a string.

06 Read directly into your world

You can use the traditional print command to read text into your game world. The following code will read the first line of your text file.

```
   mcScript = open("~/.minecraft/mcpipy/script.
txt", "r+")
  print mcScript .readline()
```

Add a few more lines to script.txt after 'Hello World'. Perhaps create some dialogue for an NPC.

07 Rehearse multiple lines at a time

You'll probably want to post several lines of dialogue into your game at a time, especially if you're creating a bot of some kind. This will post line-by-line:

```
   mcScript = open("~/.minecraft/mcpipy/script.
txt", "r+")
  print mcScript .readline()
  print mcScript .readline()
  print mcScript .readline()
```

08 Read out your script with a loop

The previous step will only read the first three lines, so you probably want to create a for loop to read the entire document line-by-line, too:

```
  for line in mcScript.readline():
```

```
print line
```

09 Write and autosave

may want to write text back to the script file:

```
with open("script.txt", "w") as mcScript:
  mcScript .write("Strings" + Variables)
```

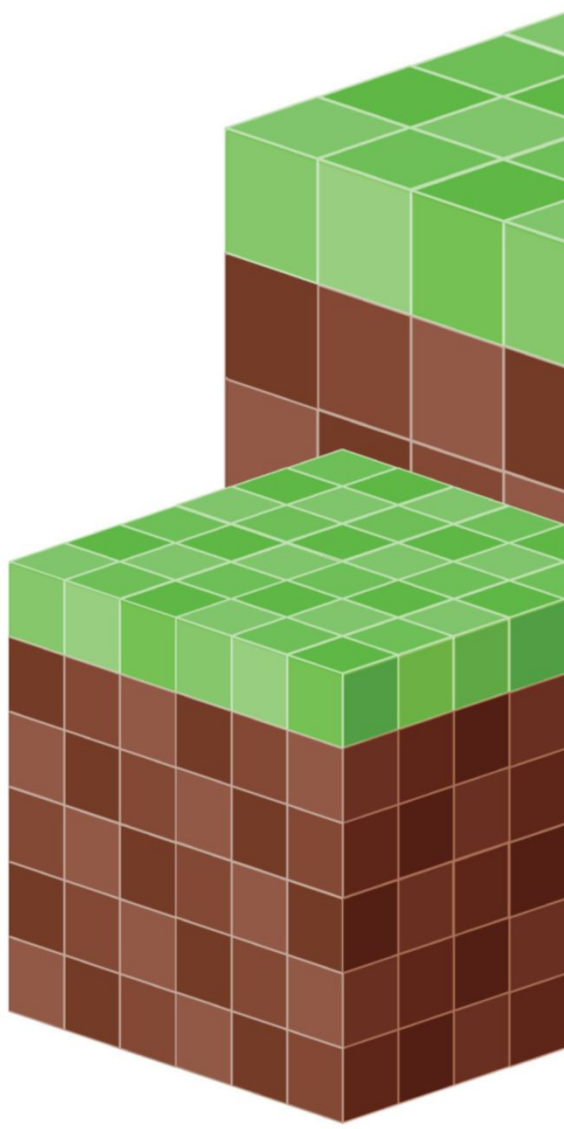
Here we can save strings and/or variables directly into the text file. If you're using with, you don't need to remember to mcScript.close() – it will save automatically.

10 Combine chat bots with previous projects

Where this gets really interesting is when you use text scripts for commands instead of dialogue. You could call a text file from a pixelArt variable instead of hard-writing numbers into the code.

```
pixelArt = mcScript .readline()
```

You can now change the pixel art every time by adjusting a line in the text file, without altering the Python code.



```
32
33  for row in range(len(pixelArt)):
34      for pixel in range(len(pixelArt[row])):
35          if pixelArt[row][pixel] == 0:
36              mc.setBlock(x, y - row, z + pixel, woolBlockOrange, woolBlockOrangeType)
37          elif pixelArt[row][pixel] == 1:
38              mc.setBlock(x, y - row, z + pixel, woolBlockBlue, woolBlockBlueType)
39          elif pixelArt[row][pixel] == 2:
40              mc.setBlock(x, y - row, z + pixel, woolBlockBrown, woolBlockBrownType)
41          elif pixelArt[row][pixel] == 3:
42              mc.setBlock(x, y - row, z + pixel, woolBlockWhite, woolBlockWhiteType)
43          elif pixelArt[row][pixel] == 4:
44              mc.setBlock(x, y - row, z + pixel, woolBlockPink, woolBlockPinkType)
```

Sphaera weather forecaster

Weather forecasting gains a magical edge with
a globe that predicts the weather by a touch of the hand

Sphaera is a deceptively simple way to display the weather forecast for the next twelve hours (using the OpenWeatherMap API) by touching five photoresistors positioned evenly around a crystal ball.

The design of Sphaera is meant to blend in with a home environment – it's something you might expect to see in the hallway that you can interact with before leaving the house.

What was the inspiration behind Sphaera?

Sphaera was made by myself and two friends. We were particularly inspired by the idea of a crystal ball and the basic idea of looking into the future. We spent some time figuring out what kind of future events we wanted to visualise, and realised that the weather was a perfect match, since it's easy to fetch from open APIs and it would look nice to project different weather states as video holograms. Since we really liked the feeling of the glass surface, we decided to not place any sensors or other material on the ball itself, but instead put everything inside.

What was your approach to the interactive design?

We wanted the interaction with the ball to be as close as possible to how you interact with a classic 'crystal ball', which means placing your hands above or around the ball in order to interact with it. This was intended to give the user an idea of how to use the device without any instructions, but it turned out that it was quite difficult to achieve. Even if the user covers

Jenny Hanell is a Masters student in Human Computer Interaction from Stockholm, Sweden. Beside her studies, she likes to develop Android apps, as well as hiking in the forest with her dog. Jenny is very interested in the psychological part of HCI and how technology affects our behaviour.

one of the light sensors, it is difficult to implement what actually happens without ruining the design or the magical feeling.

So what was your solution?

We tried many different designs. Since the holograms require darkness in order to be visible, we started out by giving the glass globe a dark hood, with the sensors hidden inside the fabric. This made it look like a little character from The Lord of the Rings or some other fantasy story, and even though it was quite amusing, that kind of ruined the elegant feeling that we wanted. So we decided to iterate a little bit, get rid of the hood and paint the inside black to achieve the required darkness.

What was the most challenging part of the project?

Definitely working on the aesthetic details. This involved placing all the sensors inside the globe and keeping them in place using strong glue; pulling down the conductive threads without having them touch each other; painting half of the globe black on the inside; and finally placing a piece of plastic inside the globe at the perfect tilted position, to reflect the holograms.

What made you decide to use the Raspberry Pi?

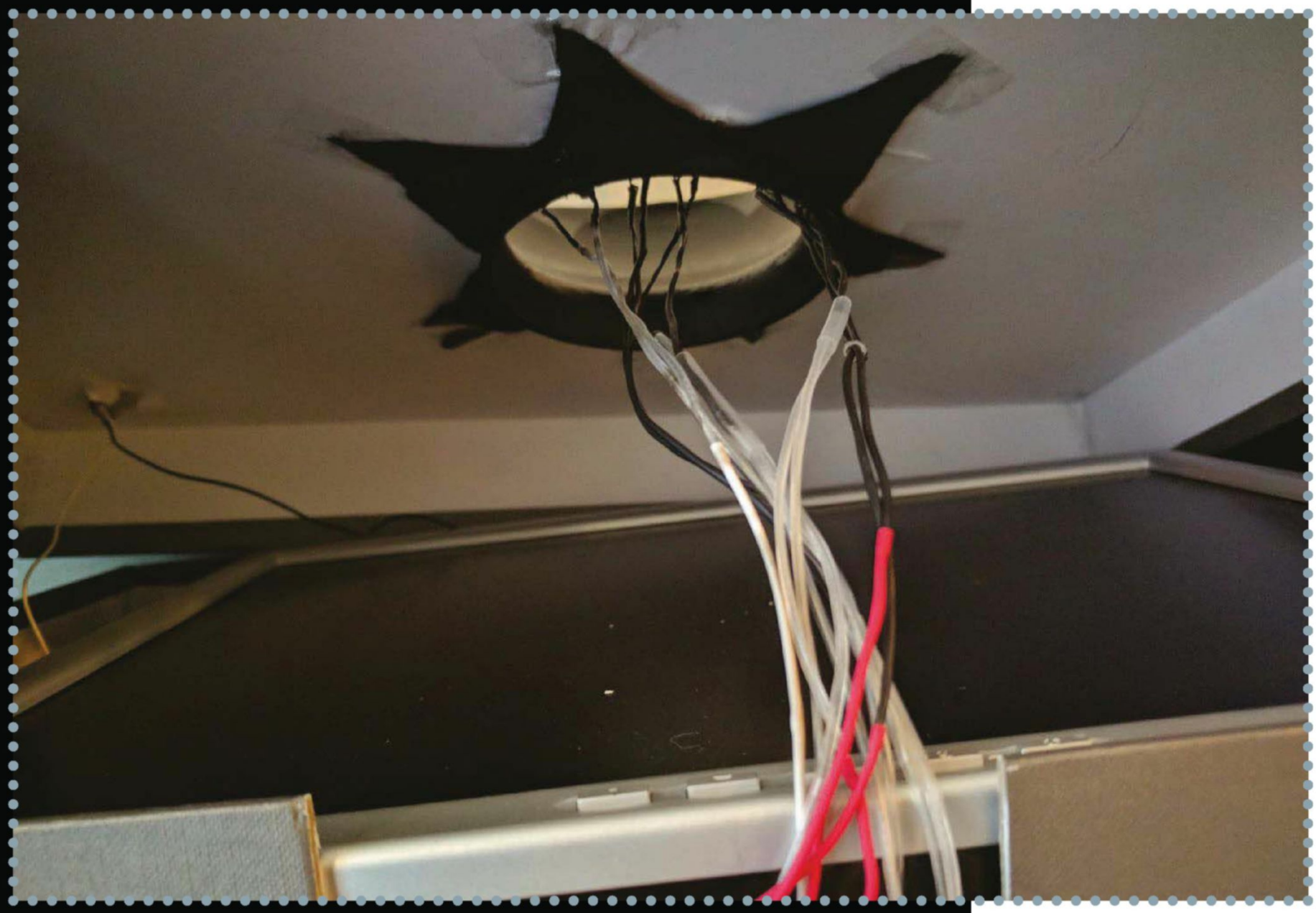
Since we were all already familiar with Arduino, we wanted to try something new. Also, the Raspberry Pi had everything we needed for this project, such as built-in Wi-Fi and Bluetooth. It was also nice that we could save some money and re-use the screen to play

THE PROJECT ESSENTIALS

- Raspberry Pi 3 (model B) plus keyboard, mouse and microSD card
- Glass globe
- A round piece of soft plastic, size depending on the diameter of the globe
- Fabric (about 1mx1m)
- LCD screen, HDMI cable and adaptor (DVI/VGA)
- 5 CdS photocells
- 4 1uf capacitors
- 1 push button
- Breadboard, chords and heat-shrink tubes
- Conductive thread (10m)
- 9 pieces of black sponge (2x1cm)
- A cardboard box big enough to fit the screen
- Scissors
- Cellplast
- Bluetooth speaker

the hologram videos instead of buying a new one. However, that made the whole device slightly large and clumsy, and a smaller screen would probably be better if anyone else wants to try the project.

Left Each photoresistor was placed inside a black sponge, with the top pointing upwards and the legs horizontal towards one of the short sides. Conductive thread was cut and wired around each photoresistor leg, which was glued inside the glass ball. Hanell describes the whole process as very difficult because the thread was so thin and she had to ensure the threads were spread out but didn't touch. She also faced problems with conductivity.



What would you do differently in hindsight?

Use less attractive but more practical solutions. For example, we used a thin and almost invisible conductive thread that went from each sensor's legs down to the breadboard. The reason for that was to avoid any ugly visible wires, but it was actually very difficult to attach it to the sensors and to make sure we didn't break the thread's conductivity while gluing and painting it. A normal slim black wire would probably work fine.

Do you have any features you want to add?

It would be nice to visualise more weather data such as temperature, and also to show the weather forecast for the whole week, not just for the next 12

Above The whole project is controlled by a Raspberry Pi 3, which is connected to both the photoresistors via a breadboard and to the LCD screen. The Pi runs Python 3 code (which you can find at http://bit.ly/SpaeraGitHub) that accesses the OpenWeatherMAP API via Wi-Fi and determines the relevant weather MP4 file to play on the screen. This is then displayed inside the ball as a 'floating' animation.

hours. It would also be cool to add speech recognition so that you can ask the globe about the weather. And of course more sensitivity – right now it only projects the standard weather conditions, but it would be nice to have different kinds of rain states, for example.

What kind of projects interest you?

I like projects that are relatively simple to start working on and don't require too much knowledge. I like to learn by doing and to jump into new projects of different character. But I really enjoy working with visualisations in different forms – to take data that is already available, put it in a new context and make it attractive and interesting somehow.

What are you hoping to do next?

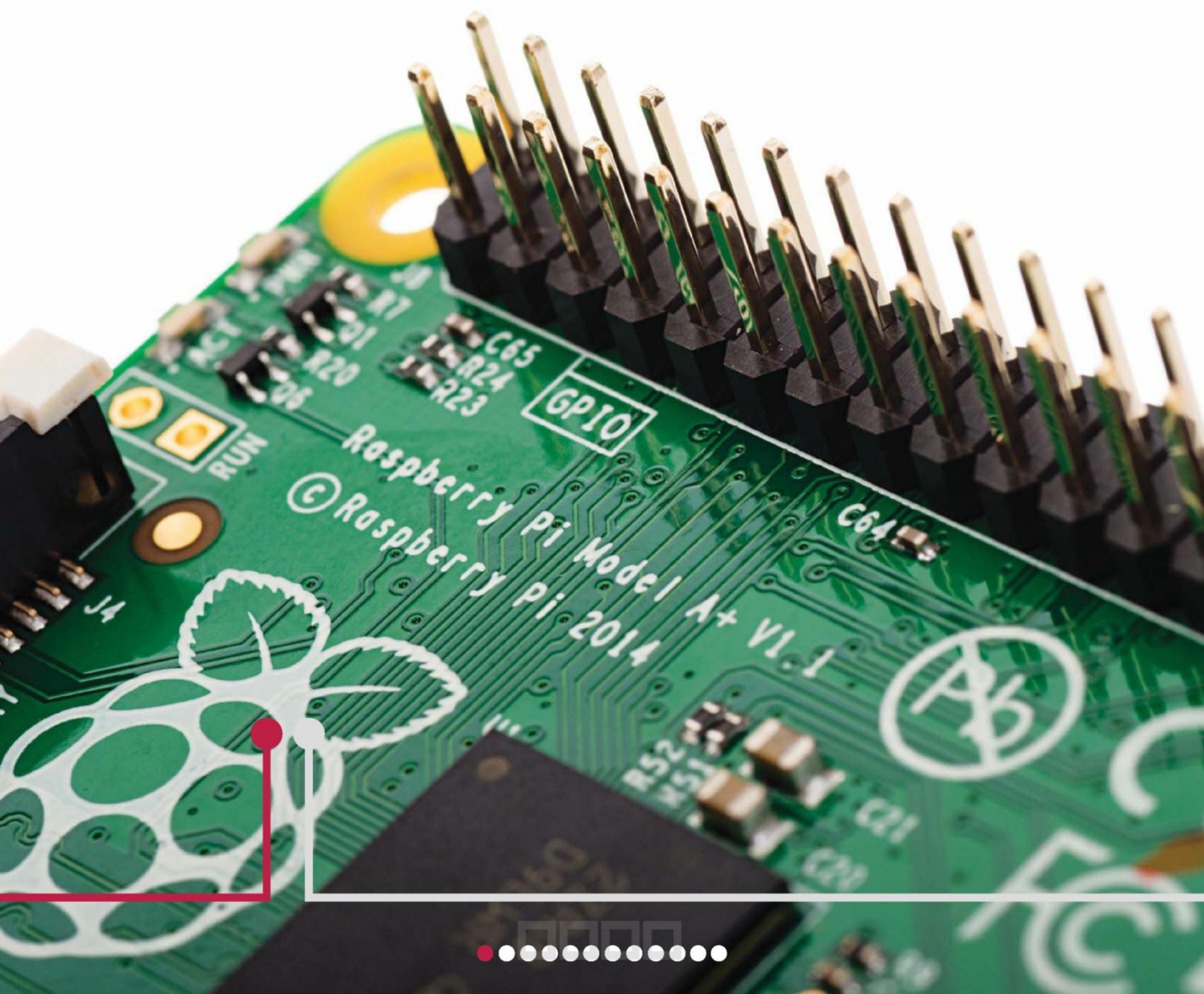
My next project will probably be something related to my toddler's room. I have some ideas about an interactive lullaby lamp or some kind of playful painting with moving animals. I'm also curious to use the Raspberry Pi in combination with Google's Android Things – it looks fun!

Further reading

To learn how to make your own Sphaera head to https://www.instructables.com/id/SPHAERA. You can download the video files for the holograms at: http://bit.ly/SphaeraVideoFiles and the code here: http://bit.ly/SpaeraGitHub.

Unlock the potential of GPIO pins with the GPIO Zero library

Make using the GPIO pins easy, fun and expand your interaction with a wide range of components and sensors



Most of us will probably remember the first time we wrote a program to light up an LED, control a motor or took a temperature reading. Nowadays this hardware and these components are relatively cheap – you can purchase a suite of sensors for the equivalent of the original cost of buying one. Components and sensors have shrunk in size with many now being etched into small add-on boards.

The libraries to control the hardware has also evolved and with this change comes the amazing GPIO Zero Python library, created by Ben Nuttall and Dave Jones, which provides a simple interface to the Raspberry Pi GPIO pins. Consider the basic example of making an LED blink: a traditional RPi.GPIO program requires 16 lines of code. With GPIO Zero you can achieve the same outcome with only five.

But don't be fooled into thinking that GPIO Zero is dumbed-down coding; GPIO Zero supports a wide range of components, sensors and parts. In this tutorial we will begin by trying out a few basic recipes before moving onto the internal devices. We'll also create a simple app to control an LED via your phone or tablet.

THE PROJECT ESSENTIALS

Raspberry Pi

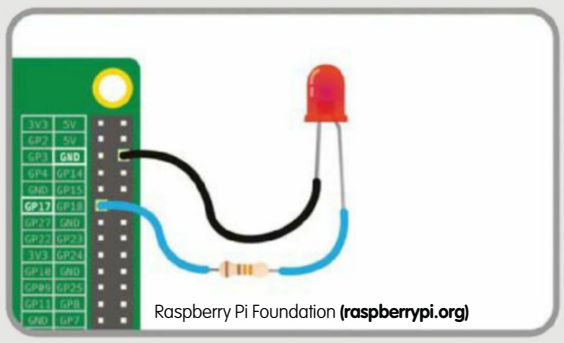
LED

Push button

Jumper wires (female to female)

01 Wire up the LED

Let's introduce the GPIO Zero library with a basic LED control program. As the tutorial progresses and we introduce more complex features we'll use the LED as the responding action, so this

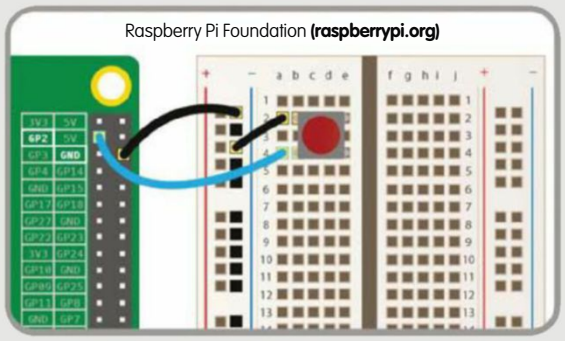


Raspberry Pi Foundation **(raspberrypi.org)**

will come in handy. Start by wrapping a suitable resistor around the positive leg of the LED and then attach the jumper wires to the two legs. The positive wire connects to GPIO pin 17 and the other wire to a ground (GND) pin.

02 Flash that LED

Now let's write simple code to flash the LED on and off. Import 'LED' from the GPIO library and assign the GPIO pin where the LED is attached – in this case, GPIO 17.



Then set up a while loop to turn the LED on for one second and then off. Save the program and run it. How easy is that?

```
from gpiozero import LED
from time import sleep

led = LED(17)

while True:
    led.on()
    sleep(1)
    led.off()
    sleep(1)
```

03 Wire up the button

Now let's wire up a button. Take the breadboard and attach the button into the holes, leaving space to connect the wires. Connect the first wire (blue) to GPIO pin 2; this provides the current for the circuit. Attach the other end

to the top leg of the button. The second wire connects to a ground (GND) pin and completes the circuit. This example uses physical pin number six, but any of the other GND pins are suitable.

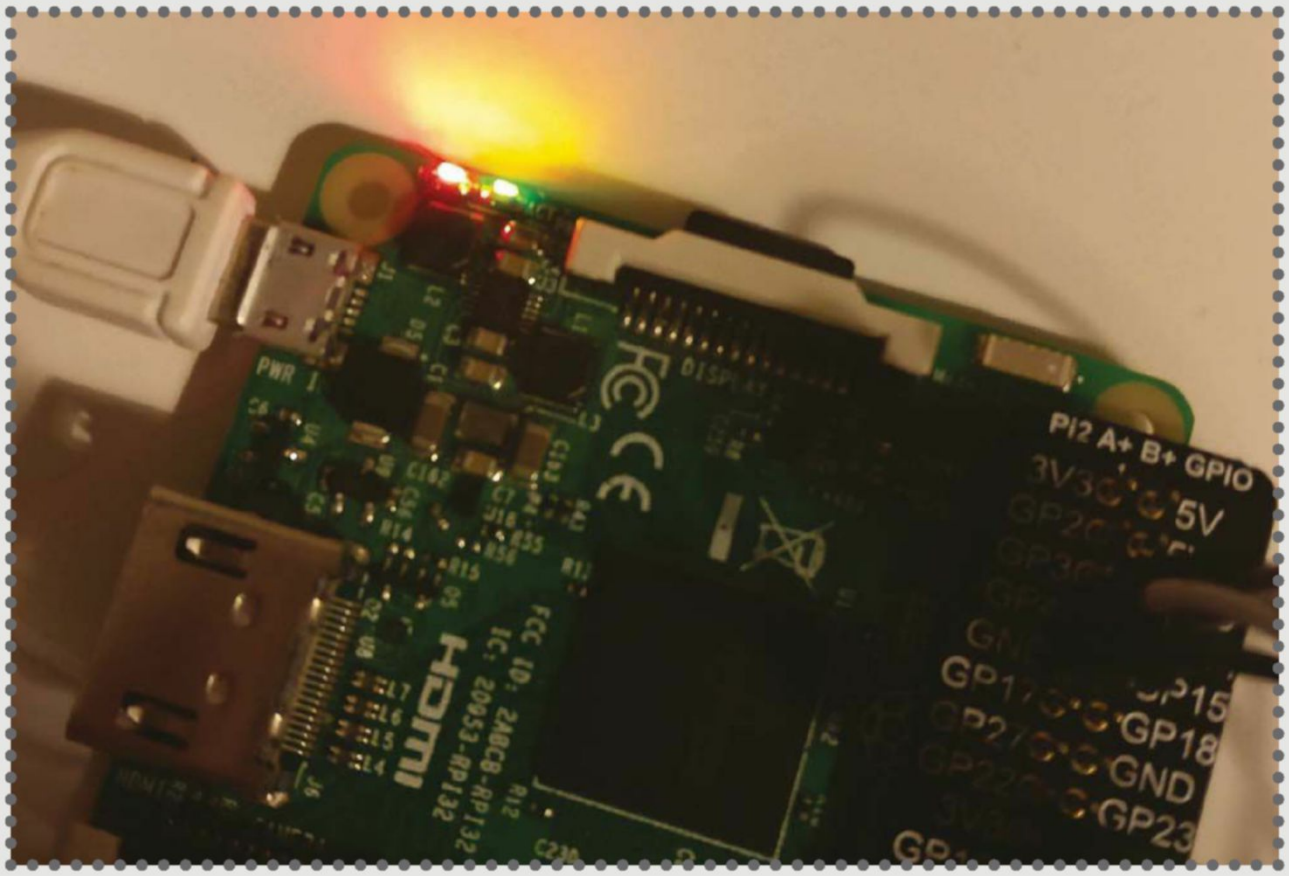
04 Code the button presses

This is just as easy – it's a simple program to control the reaction to the button being pressed. There are several 'button' programs. Our first example assigns the GPIO pin for the button on line two, then waits for a button to be pressed on line three and when pressed prints a message, on line four. In the second example, we create a function which is then called each time the button is pressed. This enables us to create more complex outcomes and actions.

```
from gpiozero import Button
button = Button(2)
button.wait_for_press()
print("Button was pressed")
```

Or somewhat more usefully:

```
from gpiozero import Button
from signal import pause

def say_hello():
    print("Hello!")

button = Button(2)

button.when_pressed = say_hello
pause()
```



05 Control the Pi's own LEDs

The GPIO Zero library makes it possible to interact and control the Raspberry Pi's built-in LEDs. These are the red power status and the green activity indicators located near the power port. Before creating your program, you need to configure each LED. Open the LX Terminal and type the following command: echo none | sudo tee /sys/class/leds/led0/trigger then press Enter. Next configure the second LED using the command echo gpio | sudo tee /sys/class/leds/led1/trigger. Now open your Python file and create the program to control them.

06 Write the program

This is another simple bit of code that flashes the activity LED followed by the power LED, then pauses.

```
from gpiozero import LED
from signal import pause
```

"The GPIO Zero library makes it possible to interact and control the Raspberry Pi's built-in LEDs"

```
power = LED(35)
activity = LED(47)

activity.blink()
power.blink()
pause()
```

07 Reset the LEDs

If you need to return the LEDs back to their orginal state – to use them for their original power and activity indication purposes – in the Terminal window type echo none | sudo tee /sys/class/leds/led0/trigger and then echo gpio | sudo tee /sys/class/leds/led1/trigger. Once you have completed this, reboot your Raspberry Pi by typing sudo reboot and then press Enter.

```
pi@raspberrypi:~ $ pinout
.--------------------------------.
|  oooooooooooooooooooo J8       |
|  1ooooooooooooooooooo          |
|                                |
|      Pi Model 3B V1.2          |
|      +----+                    |
|      |SoC |                    |
|  |D|  |    |                    |
|  |S|  +----+                    |
|  |I|                            |
|                  |C|           |
|                  |S|           |
|                  |I|  |A|      |
|  |pwr|  |HDMI|      |V|      |
|  |   |  |    |      |   |      |
'--------------------------------'

Revision           : a02082
SoC                : BCM2837
RAM                : 1024Mb
Storage            : MicroSD
USB ports          : 4 (excluding power)
Ethernet ports     : 1
Wi-fi              : True
Bluetooth          : True
Camera ports (CSI) : 1
Display ports (DSI): 1

J8:
   3V3  (1) (2)  5V
```

08 Use the pin-out tool

When using the GPIO pins it is essential to have a pin reference guide to hand. Many of us will have a scrap of paper, a neat poster or a website for this. GPIO Zero comes installed with an extremely handy board diagram utility. Open the Terminal window, type pin out and you will be presented with a graphical layout diagram. It also provides additional details about the status of the Wi-Fi, Bluetooth and other ports.

09 Install Blue Dot

Blue Dot is a super-simple app which enables you to interact with LEDs, motors and other components via a large blue dot on your phone or device. In this example we will use it to control the LED from step one. To start, open the LX Terminal window and install the Blue Dot software: type sudo apt-get install python3-dbus and then sudo pip3 install bluedot. To upgrade and add additional features, use the command sudo pip3 install bluedot upgrade.

```
File  Edit  Tabs  Help
pi@raspberrypi:~ $ sudo apt-get install python3-dbus
Reading package lists... Done
Building dependency tree
Reading state information... Done
python3-dbus is already the newest version (1.2.4-1).
python3-dbus set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
pi@raspberrypi:~ $ sudo pip3 install bluedot
```

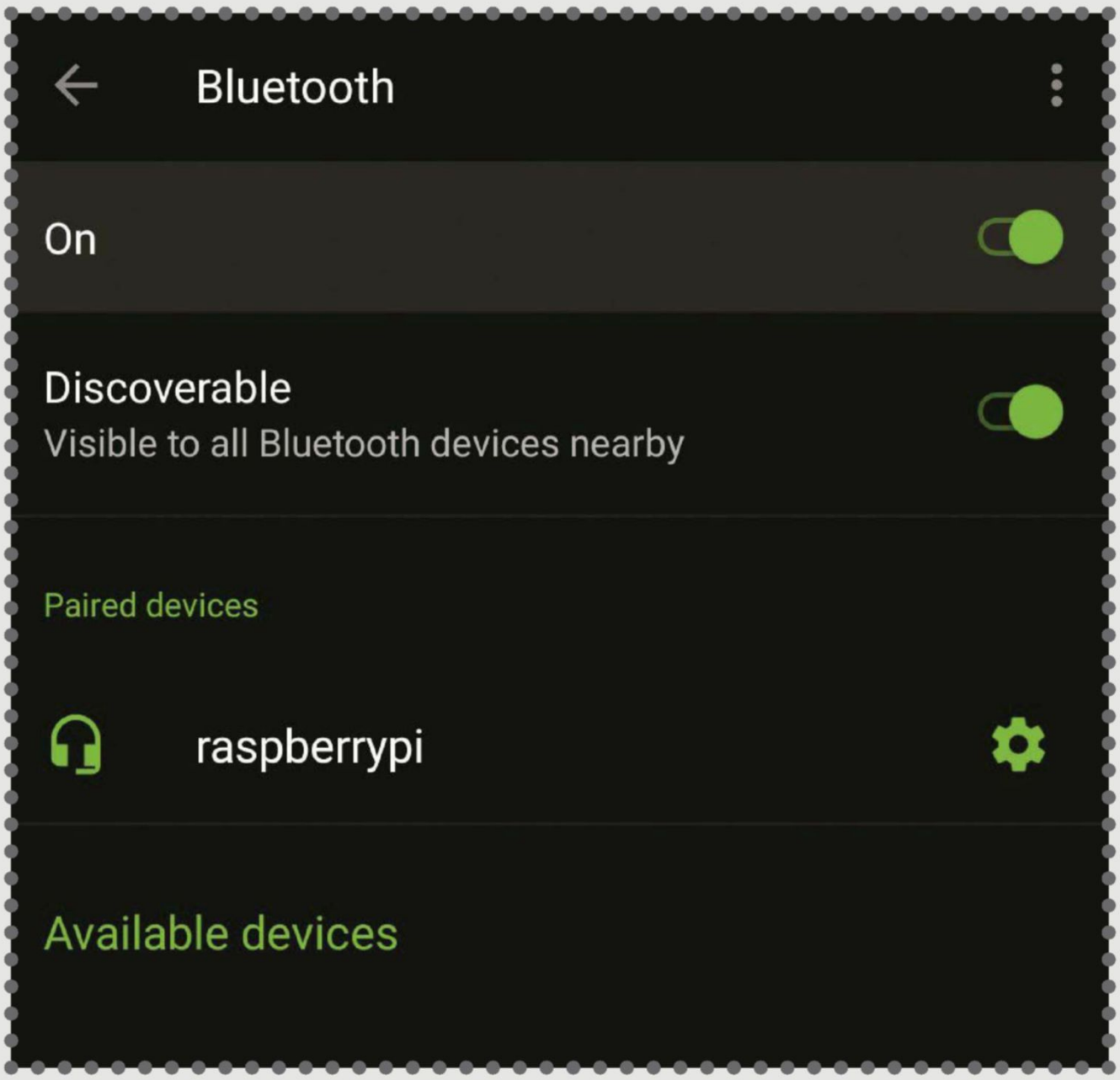
10 Download Android Blue Dot app

While the Python library is installing, head over to the Android Play Store and install the Blue Dot app. Next,

turn on your phone's Bluetooth and ensure that the discovery mode is set to Discoverable. This ensures that the Raspberry Pi can locate your device.

11 Pair your Pi and device

Now to pair your device with your Raspberry Pi. Locate the Bluetooth symbol at the top-right of the desktop. Right click the symbol and select 'Turn on' and 'Make discoverable'. You should see your device listed: select it and connect to it. You may be required to enter a shared pin number, depending on which device you're using. An alternative method to connect is to pair via your phone; search for nearby devices and then select your Raspberry Pi from the list.



"You may be required to enter a shared pin number depending on which device you're using"

12 Create your program

Start a new Python file and import the Blue Dot module (line one) and LED (line two). Next we create variables to hold the Blue Dot commands and to hold the GPIO number of the LED. On line five, we use a while loop, in which we check if the Blue Dot button on your app has been pressed, line six. If it has, we turn the LED on. When it is released turn the LED off – line nine. Ensure that your Pi and device are still paired and then run your program.

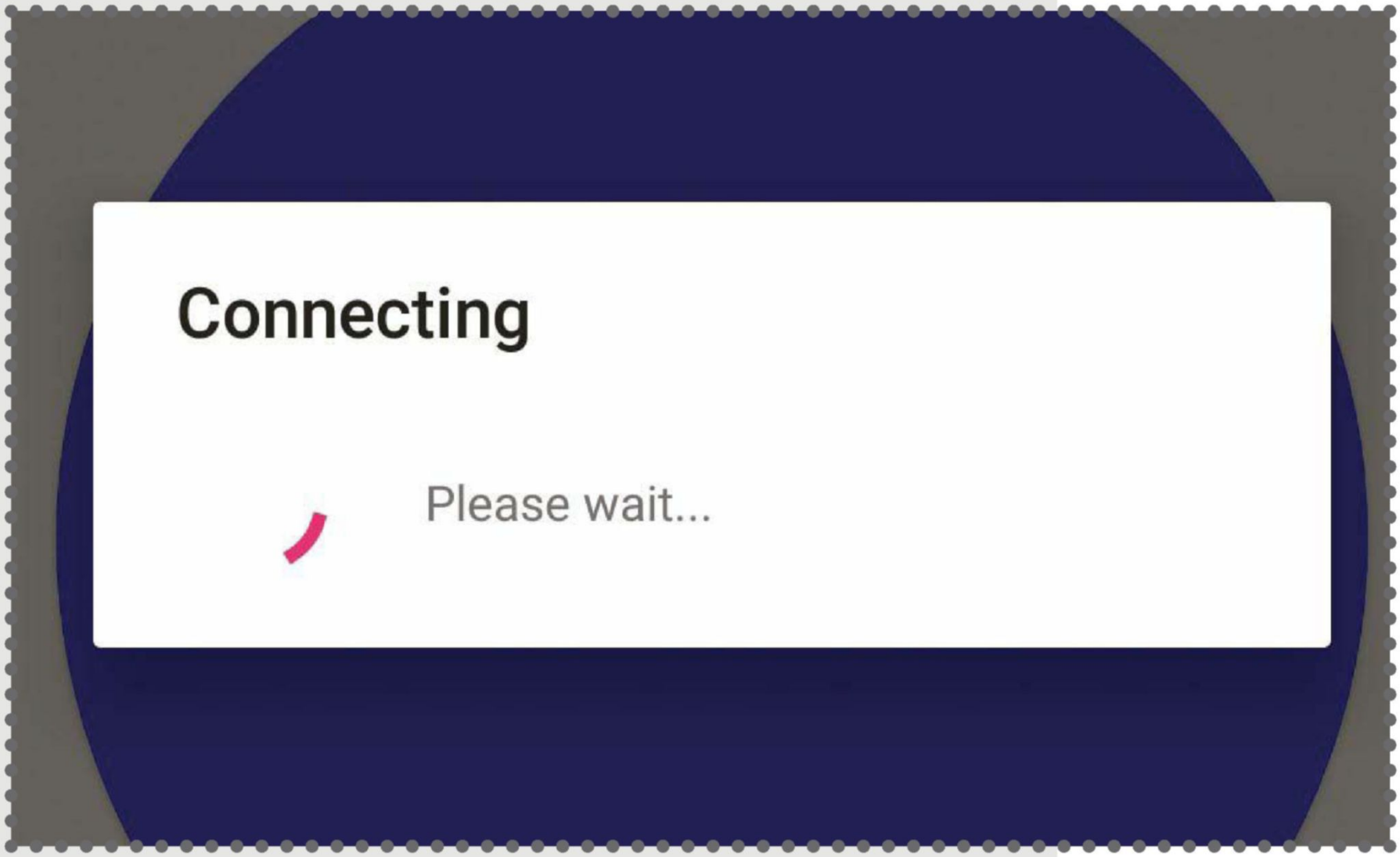
```
from bluedot import BlueDot
from gpiozero import LED

bd = BlueDot()
led = LED(17)

while True:
    bd.wait_for_press()
    led.on()
    bd.wait_for_release()
    led.off()
```

13 Run the program

With the program running and the Pi and phone paired, open the Blue Dot app on your device. From the list select your Pi and establish a connection; you'll see a confirmation message. To turn the LED on press the dot; when you release it the LED will turn off. You can use the dot to control other outputs such as motors and buzzers – check out the GPIO Zero website for more recipes.

Remote GPIO

The GPIO Zero library also includes a feature that enables you to control the GPIO pins from other devices. One of the pin libraries supported, pigpio, provides the ability to control pins remotely over the network. This means that you use GPIO Zero to control devices connected to a Raspberry Pi on the network. You can trigger the LED on and off from another computer, turn on a motor or return the reading from a sensor. Check out more details here: https://gpiozero.readthedocs.io/en/stable/recipes_remote_gpio.html



14 Ping a server

In this example we'll ping a server and check its status as online or offline. Using the same wiring as step one, in the Python file we import PingServer and LED on line one, and 'pause' on line two. Replace google.com with another address on line four. On line five, we add a 60-second delay between checks and then if the server is online, we turn the led on, line six. If the LED goes off, you know the site or server is down.

```
from gpiozero import PingServer, LED
from signal import pause
led = LED(17)

google = PingServer('google.com')

```

```
    led.source_delay = 60  # check once per
     minute
    led.source = google.values

    pause()
```

15 Check Pi CPU temperature

GPIO Zero's CPU Temperature module enables you to use an LED as a simple warning light when the Pi's CPU reaches a certain temperature. Import the required CPUTemperature, LED, pause and time modules, line one. Create a while loop and query the temperature, line six – you can set the values to check between a specific range. On line nine we use an if statement to see if the CPU temperature is greater than 60 degrees; a higher temperature will result in the LED being turned on. Alter the values as you like and run your program.

```
    from gpiozero import CPUTemperature, LED
    from signal import pause
    import time

    hot = LED(17)

    while True:
        cpu = CPUTemperature(min_temp=50, max_
     temp=90)
        print('Initial temperature: {}
     C'.format(cpu.temperature))
        time.sleep(1)

        if cpu.temperature > 60:
```

```
            hot.on()
        else:
            print (“Cool!”)
```

16 React to time of day

This final program takes a time reading from the Pi's internal clock and responds by turning on the LED when the time is within a specific range. First we import the TimeOfDay and LED modules, line one, then the time, line two, and again 'pause' on line three. Set your specific time range on line five – in this example between the hours of 7am and 9am. You may need to adjust these values if you are testing after 9am. Finally, set the LED to turn on when the time meets the morning values, between 7am and 9am.

```
from gpiozero import TimeOfDay, LED
from datetime import time
from signal import pause

light = LED(17)

morning = TimeOfDay(time(7), time(9))
light.source = morning.values

pause()
```

Read and return the orientation of an object

Set up and code a tilt switch to read its orientation and respond with a desired action

A tilt switch is a component capable of sensing a change in orientation in four different directions: up, down, left and right. The tilt switch from Design Spark makes it easy to incorporate this sensor into your projects by breaking the necessary pins out to standard 0.1" spaced headers. The board works at either 3.3 or 5 volts making it perfect for use with a Raspberry Pi. You can use it to orientate a robot, checking to see if it has fallen over, or how about embedding it within a box to check that it is facing the correct way up? Although not originally designed for the Raspberry Pi, this tutorial shows you how to set the switch up and combine it with the GPIO pins. It also covers a simple program to read the orientations and respond, meaning you can deploy it in your own Raspberry Pi projects.

01 Set up and solder wires

The tilt switch works using a small piece of mercury or a ball bearing, which joins any two of the inputs. When a small current is passed through the tilt switch, an

electrical circuit is either made or broken at one side of the switch. Reading where this occurs means the orientation of the switch can be measured. Basically, it works as if two of the inputs are buttons. You can press both buttons at the same time, or press none of the buttons. One button is pressed and the other is not, or the other button is pressed and the opposite is not. This gives you four possible states that the buttons can be in and therefore, four different options. This allows you to measure up to four orientations. Solder the pin mounts into place and leave the connections to cool down.

02 Connect to the GPIO pins

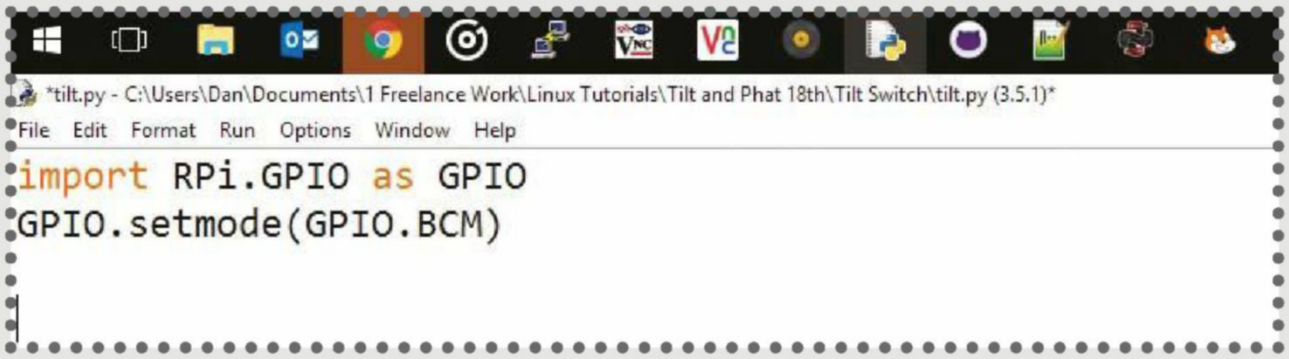
Ensuring that your Raspberry Pi is switched off, take your four female-to-female jumper jerky wires. Place the first wire onto the 3.3volts pin; this is the first physical pin on the board. Connect the free end to the VCC pin on the tilt switch. Take another wire and connect it to a Ground pin. These are found at physical pins 6, 9, 14, 20, 25, 30, 34, 39. Connect the other end to the GND pin on the switch. Follow the same method and connect the last two wires to pins GPIO 4 and 25. These are physical pins 7 and 22. Attach the other ends to the appropriate connection on the tilt switch. GPIO 4 connects to S1 and GPIO 25 to S2.

```
sudo apt-get update
sudo apt-get dist-upgrade
```

The upgrade process will take some time. You'll see a message about the plymouth I/O multiplexing framework; press Q to dismiss this and proceed.

GPIO pin numbering

GPIO pins are a physical interface between the Pi and the outside world. At the simplest level, you can think of them as switches that you can turn on or off. You can also program your Raspberry Pi to turn them on or off (output). The GPIO. BCM option means that you are referring to the pins by the Broadcom SOC channel number. If you start counting the pins this is the physical pin number. The GPIO. BOARD option specifies that you are referring to the pins by the plug number, i.e. the numbers printed on the board



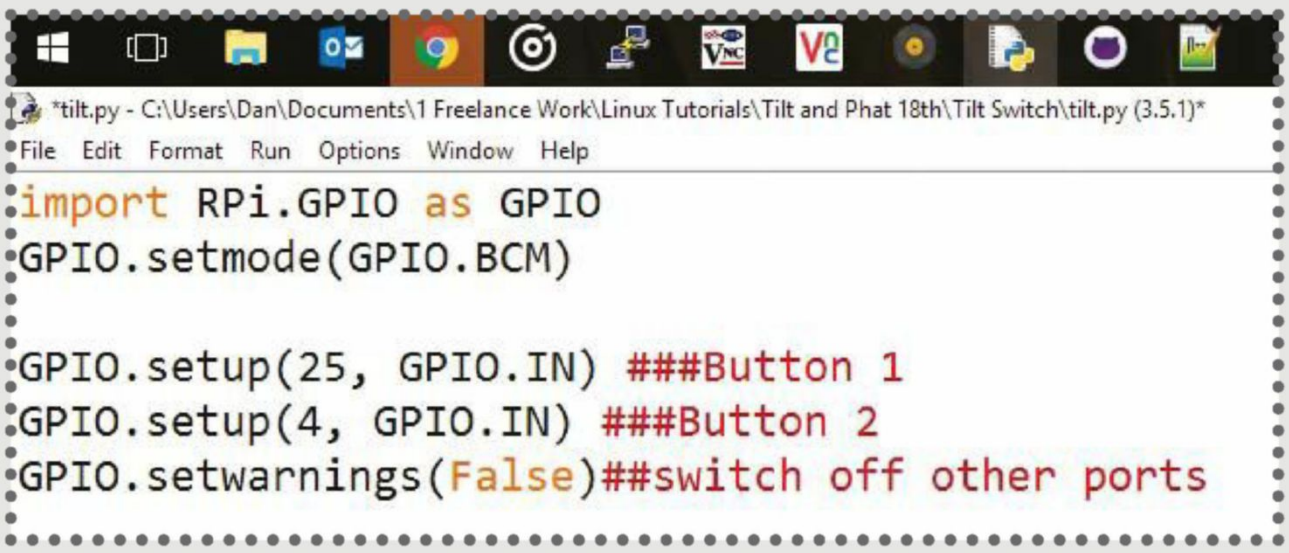
03 Create the code

Boot up your Raspberry Pi and load the LX Terminal window, type:

```
sudo idle3
```

This will load the Python 3 editor (In order to access the GPIO pins and control them you require administrator access. Using the prefix sudo grants you this). Open a New Window and at the top, import the Raspberry Pi GPIO library, line 1. Next set the GPIO mode to use the BCM layout, line 2. This sets the board to use the Broadcom pin layout.

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
```



04 Set up and read inputs

Next, set up the code to identify and read the inputs from the tilt switch. The first input is read from pin number 25,

```
*tilt.py - C:\Users\Dan\Documents\1 Freelance Work\Linux Tutorials\Tilt and Phat 18th\Tilt Switch\tilt.py (3.5.1)*
File  Edit  Format  Run  Options  Window  Help
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)

GPIO.setup(25, GPIO.IN) ###Button 1
GPIO.setup(4, GPIO.IN) ###Button 2
GPIO.setwarnings(False)##switch off other ports

while True:
    if GPIO.input(25) == 0 and GPIO.input(4) == 0:
        print ("Down")
```

line 1. Remember that you are using the BCM pin layout so this is physical pin number 22 on your Pi. On the line underneath add the second input, using the same code but setting pin 4 as the input, line 2. Finally, turn off all the other pins so that no readings are taken from these, line 3.

```
GPIO.setup(25, GPIO.IN) ###Button 1
  GPIO.setup(4, GPIO.IN) ###Button 2
  GPIO.setwarnings(False)##switch off other
    ports
```

05 Reading one: down

Now to create the code that will recognise and respond to one of the states that the switch is in. Under the previous line of code, create a while loop, line 1, then indent the next line and check for 'no input' on pins 25 and 4. This basically checks both the inputs are not receiving any current. In the button analogy, both buttons are not being pressed at the same time. Assuming that your tilt switch is set up in the same way as the tutorial, then this state corresponds to the switch being in the down position. Use a simple print statement to tell the user the current orientation, line 3.

```
  while True:
     if GPIO.input(25) == 0 and GPIO.input(4) ==
    0:
        print (“Down”)
```

06 Reading two: up

Next add a new conditional to check for another state. This is both inputs being active. An active state is denoted by a value of 1 or True. On the next line down, add an if statement to check inputs 25 and 4, line 1. This state is the ‘up’ orientation so use a print statement to notify the user, line 2. Remember to indent the second line in order to avoid a syntax error.

```
  if GPIO.input(25) == 1 and GPIO.input(4) ==
    1:
        print(“Up”)
```

07 Reading three: left

The program continues in a similar pattern, use an if statement to check the readings on pin 25 and pin 4. If the readings returned are ‘pin 25 is live and pin 4 is zero’, then the switch is orientated to the left. Add the code below, remembering to include the required indentation.

```
     if GPIO.input(25) == 1 and GPIO.input(4) ==
    0:
        print(“Left”)
```

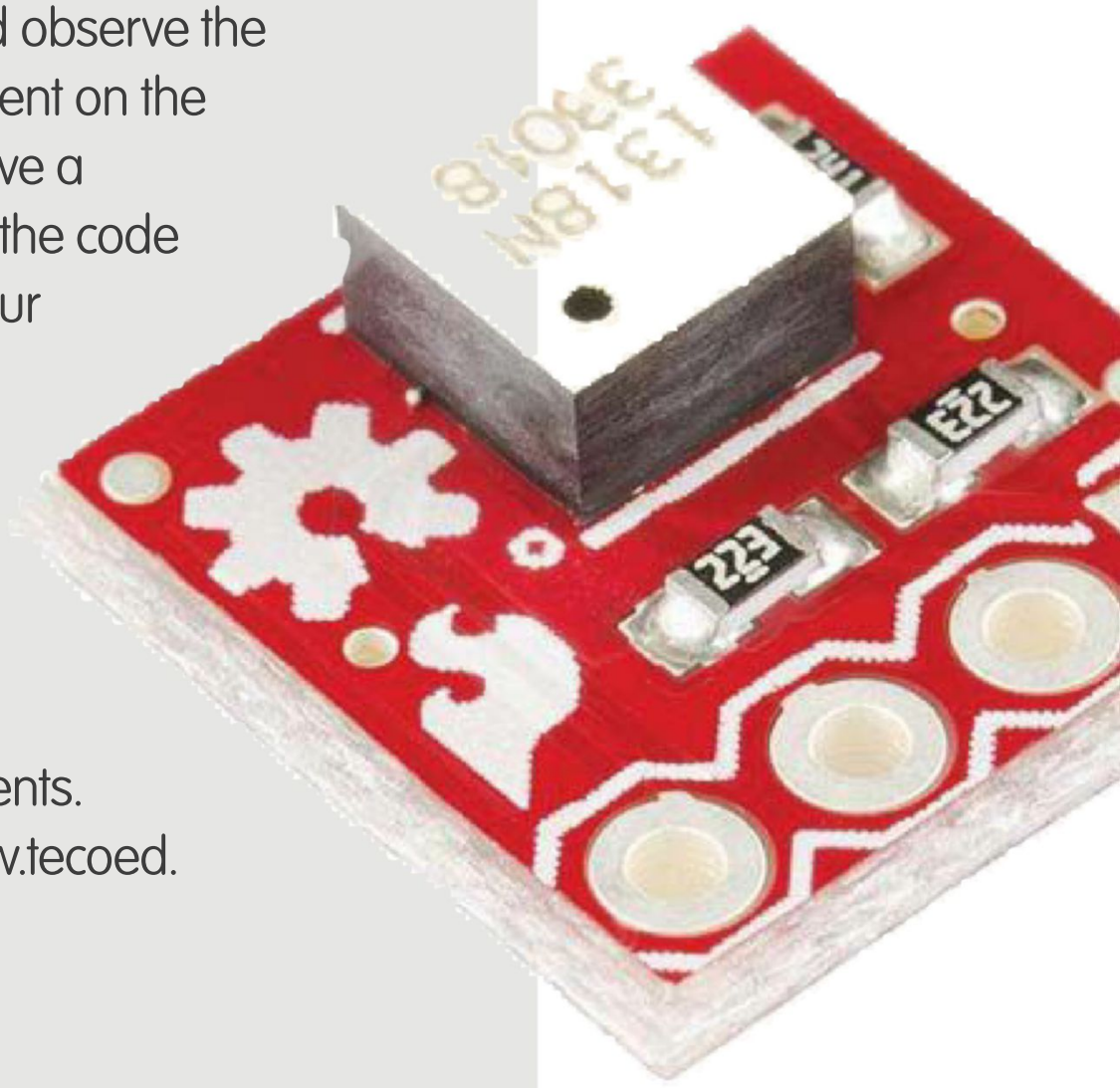
08 Reading four: right

The last step is to add the final lines of code to check if the switch is tilted to the right as it should be. Again, use an if statement, line 1 to check for ‘no value on pin 25 and a value of one on pin four’. As before, print out a message for the user.

```
    if GPIO.input(25) == 0 and GPIO.input(4) ==
      1:
          print(“Right”)
```

09 Run the program

Save your program and run it by pressing F5 on the keyboard. Rotate the tilt switch and observe the orientation and the printed statement on the console window. Now that you have a working tilt switch you can modify the code to add actions or responses for your own projects. Replace the print statements in your program with the required actions. For example, you can create a talking book that knows which direction it is facing and responds with a selection of suitable comments. See the example here, http://www.tecoed.co.uk/get-to-the-chopper.html.

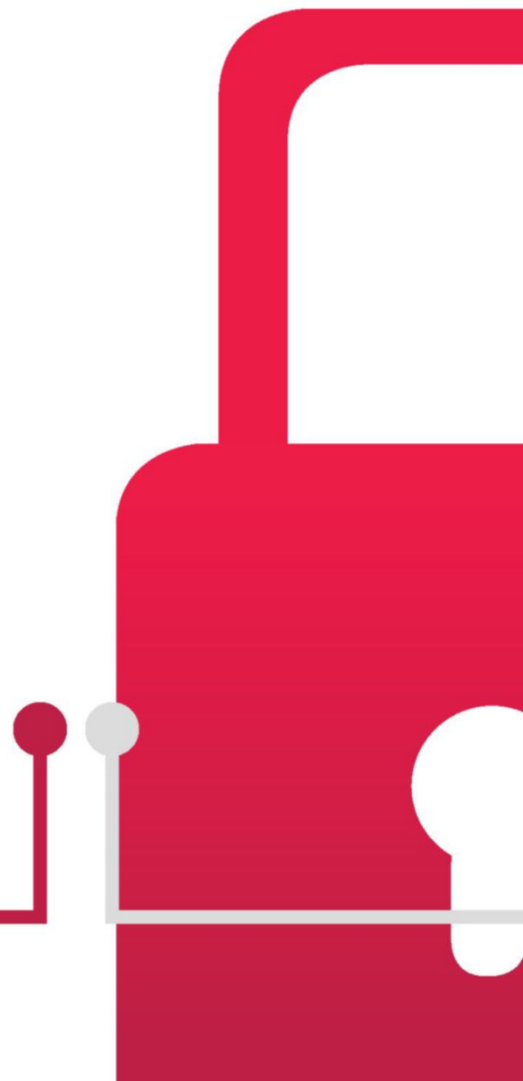


Managing data in Python

When the amount of data you need to work with goes beyond easy flat files, it's time to move into using a database and a good place to start is SQLite

In previous issues, we have looked at how to use data that is stored in files using regular file I/O. From here, we moved on to looking at how to use pandas to work with more structured data, especially in scientific work. But, what do you do when you have data that needs that go beyond these tools, especially in non-scientific domains? This is where you likely need to start looking at using a database to manage information in a better way. This month, we will start by looking at some of the lighter options available to work with simple databases.

In terms of lightweight databases, SQLite is the de facto standard in most environments. It comes as a C library that provides access to a file-backed database that is stored on the local disk. One huge advantage is that it does not need to run a server process to manage the database. All of the code is actually part of your code. The query language used is a variant of standard SQL. This means that you can start your project using SQLite, and then be able to move to a larger database system with minimal changes to your



code.

There is a port to Python available in the module 'sqlite3' which supports all of the functionality. Because it is the standard for really lightweight database functionality, it is included as part of the standard Python library. So you should have it available wherever you have Python installed. The very first step is to create a connection object that starts up the SQLite infrastructure:

```
import sqlite3
my_conn = sqlite3.connection('example.db')
```

This gives you a connection object that enables interactions with the database stored in the example.db file in the current directory. If it doesn't already exist, the sqlite3 module will create a new database file. If you only require a temporary database that needs to live for the duration of your program run, you can give the connection method the special filename ':memory:' to create a database stored solely in RAM.

Now that you have a database, what can you do with it? The first step is to create a cursor object for the database to handle SQL statements being applied to the database. You can do so with my_cursor = my_conn.cursor().

The first database thing you will need to do is to create tables to store your data. As an example, the following code creates a small table to store names and phone numbers.

```
my_cursor.execute('''CREATE TABLE phone
(name text, phone_num text)''')
```

You have to include the data type for each column of the table. SQLite natively supports SQL data types BLOB, TEXT, REAL, INTEGER and NULL. These map to the Python data types byte, str, float, int and None. The execute method runs any single SQL statement that you need to have run against the database. These statements are not committed to the file store for the database, however. In order to have the results actually written out, you need to run my_conn.commit(). Note that this method is part of the connection object, not the cursor object. If you have a different thread also using the same SQLite database file, it won't see any changes until a commit is called. This means that you can use the rollback() method to undo any changes, back to the last time commit() was called. This allows you to have you a rudimentary form of transactions, similar to the functionality of larger relational databases.

Now that we have a table, we should start populating it with data. The simplest way to do this is to use a direct INSERT statement, e.g.:

```
my_cursor.execute("INSERT INTO phone VALUES
('Joey Bernard', '555-5555')")
```

While this is okay for hard-coded values, you'll probably have data coming from the user that needs to be entered into the database. In these cases, you should always check this input and sanitise it so there's no code that can be used for an SQL injection attack. You can do this, then do string manipulation to create the complete SQL statement before calling the

Why Python?

It's the official language of the Raspberry Pi. Read the docs at python.org/doc

execute method. The other option available is to use an SQL statement that contains placeholders that can be replaced with the values stored in variables. This makes the validation of the input data a bit easier to handle. The above example would then look like:

```
my_name = 'Joey Bernard'
my_number = '555-5555'
my_cursor.execute("INSERT INTO phone VALUES
(?,?)",
(my_name,my_number))
```

The values to be used in the SQL statement are provided within a tuple. If you have a larger amount of data that needs to be handled in one go, you can use the executemany() function, available in the cursor object. In this case, the SQL statement is structured the same as above. The second parameter is any kind of iterable object that can be used to get a sequence of values. This means that you could write a generator function if your data can be processed that way. It is another tool available to automate your data management issues.

Now that we have some data in the database, how can we pull it back out and work with it? The basic SQL statement that is used is the SELECT statement. You can use the following statement to get my phone number.

```
my_cursor.execute("SELECT phone_num FROM
phone WHERE name=:who", {"who":'Joey
Bernard'})
```

```
print(my_cursor.fetchone())
```

As you can see, you need to call some kind of fetching method in order to get your actual results back. The fetchone() method returns the next returned value from the list of returned values. When you reach the bottom of the list, it will return None. If you want to process returned values in blocks, you can use the cursor method fetchmany(size), where size is how many items to return bundled within a list. When this method runs out of items to return, it sends back an empty list. If you want to get the full collection of all items that matched your SELECT statement, you can use the fetchall() method to get a list of the entire collection. You do need to remember that any of the methods that return multiple values still start wherever the cursor currently is, not from the beginning of the returned collection.

"All of the SQLite code is part of your code"

Sometimes, you may need to add some processing functionality to the database. In these cases, you can actually create a function that can be used from within other SQL statements. For example, you could create a database function that returns the sine of some value.

```
import math
my_conn.create_function("sin", 1, math.sin)
cursor2 = my_conn.cursor()
cursor2.execute("SELECT sin(?)", (42,))
print(cursor2.fetchone())
```

There is a special class of database functions, called aggregators, that you may wish to create, too. These take a series of values and apply an aggregation

function, like summing, over all of them. You can use the create_aggregate() method to register a class to act as the new aggregator. The class needs to provide a step() method to do the aggregation calculation and a finalize() method that returns the final result.

One last item you may want to be able to do is to have larger blocks of SQL statements run against your data. In this case, you will want to use the cursor object's executescript() method. This method takes a string object that contains an entire script and runs it as a single block. One important difference here is that a commit() call is made just before your script is run. If you need to be able to do rollbacks, this is an important caveat to keep in mind. When you start to have more complicated queries, you may need to track where your results came from. The description property of the cursor object returns the column names for the last executed query.

When you are all done, you should always call the close() method of the connection object. But, be aware that a commit is not done automatically. You will need to call it yourself before closing. This ensures that all of your transactions are flushed out to disk and the database is in a correct state. Now you can add more robust data management to your code. ■

Next issue

Get inspired **Expert advice** **Easy-to-follow guides**

"Write more with Scripto"

Get this issue's source code at:

www.linuxuser.co.uk/raspicode